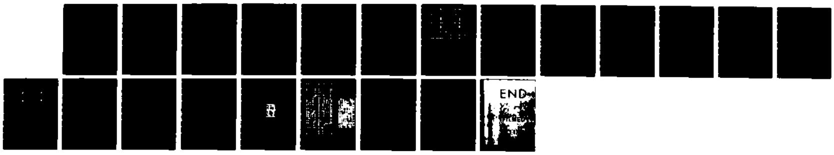
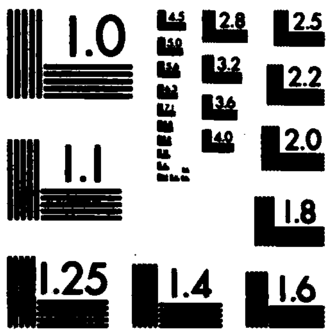
AD-A132 521   OPTIMIZATION TECHNIQUES FOR IC LAYOUT AND COMPACTION   1/1
(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE
G KEDEM ET AL. SEP 82 TR-117 N00014-82-K-0193
UNCLASSIFIED   F/G 9/5   NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

AD-A132 521

Optimization Techniques for IC Layout and Compaction

Gershon Kedem and Hiroyuki Watanabe
Computer Science Department
The University of Rochester
Rochester, NY 14627

TR117
September, 1982

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

rochester

Department of Computer Science
University of Rochester
Rochester, New York 14627

DTIC
SEP 1 6 1983
H

DTIC FILE COPY

83 09 13 128

Optimization Techniques for IC Layout and Compaction

Gershon Kedem and Hiroyuki Watanabe
Computer Science Department
The University of Rochester
Rochester, NY 14627

TR117
September, 1982

Abstract

This paper describes a new approach for IC layout and compaction. The compaction problem is translated into a mixed integer-linear programming problem of a very special form. A graph based optimization algorithm is used to solve the resulting problem. An experimental program that uses the above techniques is described. The program could be used either as an aid to hand layout or as the bottom part of an automatic layout generation program.

The preparation of this paper was supported in part by the Defense Advanced Research Projects Agency Grant No. N00014-82-K-0193, in part by ONR Grant No. N00014-80-C-0197, and in part by Defense Advanced Research Projects Agency Grant No. N00014-78-C-0164.

DTIC
COPY
INSPECTED
2

Accession For
NTIS GRA&I [✓]
DTIC TAB ☐
Unannounced ☐
Justification
By letter
Distribution/
Availability Codes

Dist	Avail and/or Special
A	

1. Introduction

One of the most time consuming and error prone parts of custom integrated circuit design is low level cell layout. The logic designer can design a topological layout of a digital circuit in the form of a "stick diagram" [Mead & Conway 1979; Williams 1977] quite rapidly. However, the translation of this design into an actual layout is tedious and error prone. Moreover, the same stick diagram can be mapped to a mask level layout in many different ways. On different occasions it is desirable to layout the same functional block differently. For example, the aspect ratio of the block might need to be changed so it would fit other parts of the design, or transistor sizes might need to be changed depending on the "fanout" load they have to drive.

Having a library of cells in stick diagram form together with a set of procedures to translate them into layout is much more flexible and therefore more useful than a standard cell library. The cells could be scaled automatically according to fan in and fan out considerations. Their shapes could be altered automatically to fit other parts. Moreover changes in design rules will not render the cells obsolete.

Several algorithms that translate stick diagrams into layout have appeared in the literature [Williams 1978; Dunlop 1978; Hsueh 1979a]. However, these algorithms are heuristic and *ad hoc*. Our approach is to translate the layout compaction problem into an optimization problem.

The compaction problem is stated as a minimization problem; that is,

find the minimum of the area function subject to linear and nonlinear constraints.

The optimization problem resulting is a mixed integer linear programming problem of a very special kind. The integer variables are {0,1} decision variables. For every fixed set of decision variables, one needs to solve two independent longest-path problems, one for the horizontal direction and the other for the vertical one. The interaction between the horizontal and vertical compaction is via the integer decision variables. More details on the problem formulation are given in Section 2.

The constraints come from geometrical design rules, from connectivity information and possibly from user specified constraints.

The ability to specify additional constraints to the problem is very useful. For example, when one designs a cell that is to be replicated many times (most popular example is the shift register cell in Mead & Conway), it is important to be able to specify that the center of an input node would be at the same height as the center of a given output node. Using our technique, it is easy to compact cells in one direction while keeping the other dimension fixed. This feature is useful when one makes cells fit one another. Moreover, by manipulating the objective function it is possible to influence the final shape of the compacted cells.

We would like to stress that our compaction scheme is truly a two-dimensional compaction. Previous methods mentioned in the literature [Williams 1978; Dunlop 1978; Hsueh 1979a] use two one-dimensional compaction schemes.

The circuit compaction problem is made of three parts:

1) Translation of stick diagrams into an optimization problem.

2) The solution of the optimization problem

3) Making modifications to the stick diagrams and mathematical optimization to try and improve the layout. For example, inserting jog points or symbol rotations.

In Section 2, we discuss the problem formulation and the translation from stick diagrams to the mathematical optimization problem. In Section 3, we describe the branch and bound technique we use to solve the optimization problem. In Section 4, we give examples and some data about the experimental program we have developed.

2. Problem Formulation and Constraint Generation

Stick diagrams are a well known symbolic form for describing integrated circuits (see [Williams 1977; Mead & Conway 1979]). Stick diagrams provide a topological description of the circuit specifying transistor size and type, the connecting lines, the layers they are made of and their connections to other lines or transistors. Loosely speaking, the layout is made of hard fixed-shape cells. Cells are pull-up and pull-down transistors, pass transistors, capacitors, and contacts. Interconnecting these cells are rubber-band like lines that could be stretched and contracted along the horizontal or vertical axis, but have a fixed width. The width of lines is either the default minimal width or larger.

In our system, the input is a CIF file. Each transistor or contact is an instance of a predefined symbol from a library. Interconnecting these cells are line segments. Each line segment is terminated either by touching a symbol or another line segment or by touching the outside boundary of the global cell. By convention, lines that are drawn with width less than or equal to the minimal width are assumed to be of minimal width and lines that are wider are assumed to be constrained to be the width that they are drawn at. All line segments are either horizontal or vertical. All vertical or horizontal line segments stay vertical or horizontal respectively, under the compaction process. Symbol orientations stay the same, too. Figure 1 shows an example of the input to the system.

Informally, the circuit compaction problem is as follows: Assume that the lower left corner of the main symbol is at the origin. Specify the centers of all cells. The horizontal coordinate of each vertical line segment, the vertical coordinate of the center of each horizontal line segment, and the length of each line segment such that all geometrical design rules are satisfied and the connectivity of the original circuit is not altered.

Geometrical design rules come in three varieties, separation rules, line-width rules, and overlap rules. Since all transistors and contacts are predesigned, overlap rules are automatically satisfied. Line-width rules are built-in; that is, the width of each line is computed and is assumed to be fixed. Line-width enters into the calculations as constants in the constraint generation part.

The compaction problem is translated into a mixed linear-integer programming problem with a nonlinear objective function, namely the area of the compacted cell.

The linear constraints come from distance and connectivity requirements between the various components of the circuit. Although the problem could be solved as a general mixed integer programming problem, doing so would be very inefficient and expensive. The problem we actually solve is formulated as a graph problem. All linear constraints are encoded as arcs on a graph. For example, the encoding of the distance requirement between a poly-silicon line and a poly-solicon-metal contact cell is shown in Figure 2.

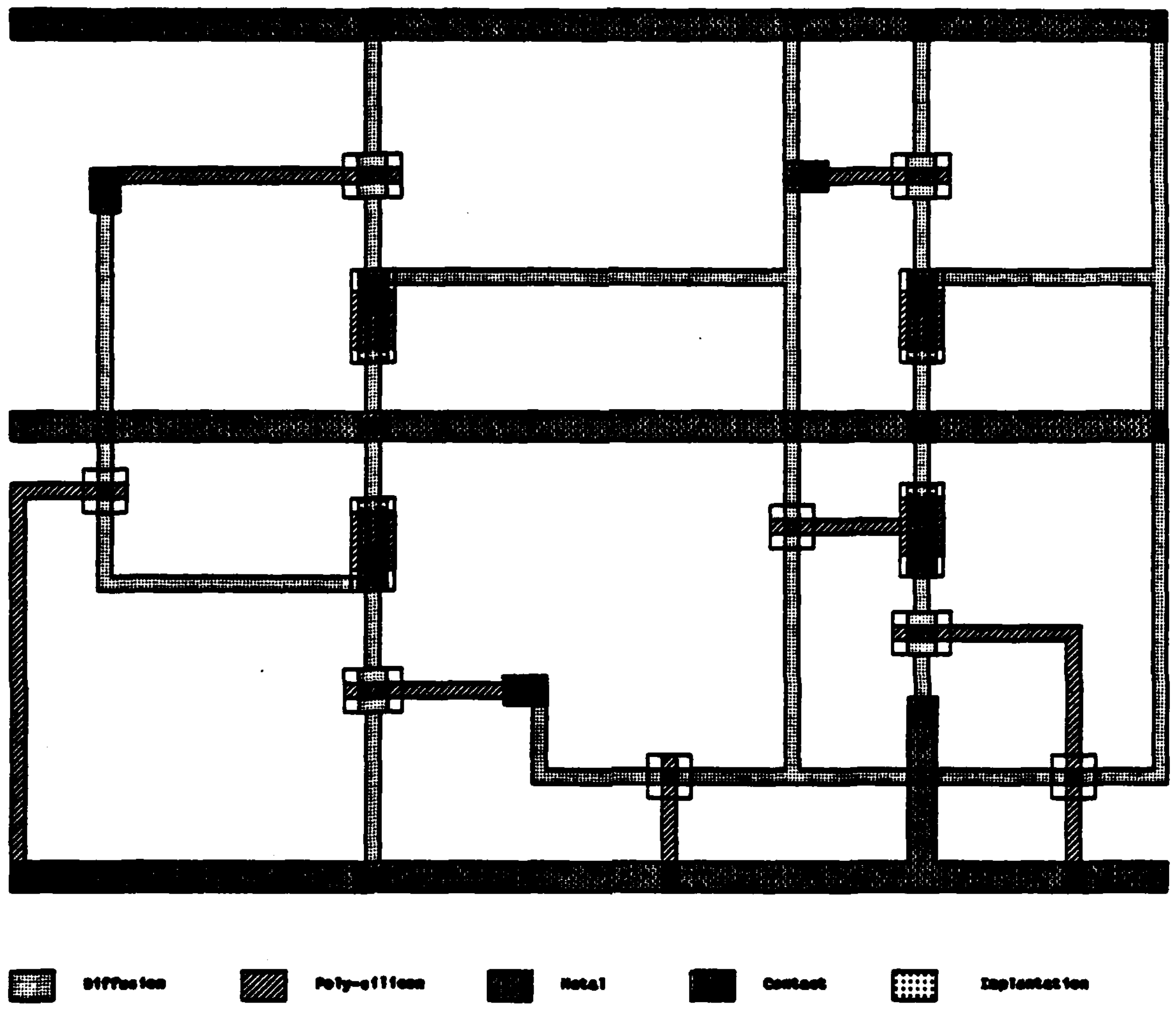


Figure 1: Stick diagram of T-flipflop

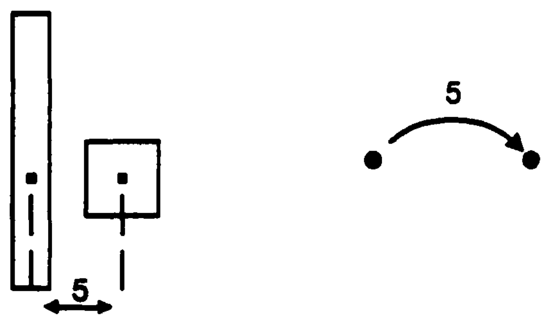


Figure 2: A simple arc constraint

Now both x_1 and x_2 are nodes in the graph with an arc with weight 5 in between. We construct two graphs, an X-graph and a Y-graph. The nodes in the X graph are the X coordinates of all the center points of the symbols and the centers of vertical lines. Similarly the nodes in the Y-graph are the Y coordinates of the symbols' centers and the horizontal lines.

The integer decision variables come from the interaction of corners. (See Figure 3.)

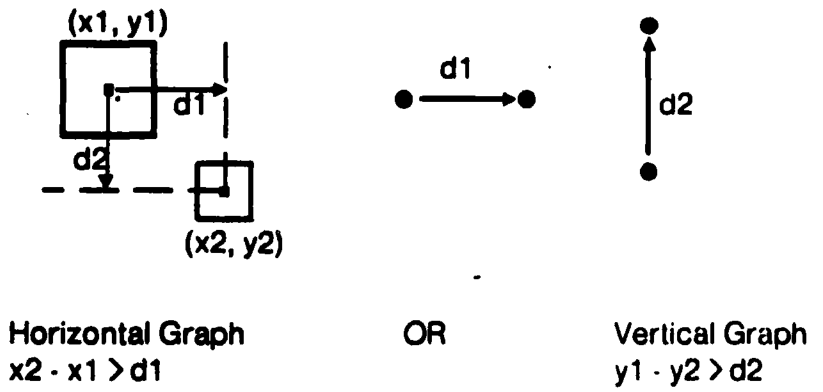


Figure 3: Dual-arc Constraints

For each pair of constraints, there is a decision variable. Its value determines whether the X arc or the Y arc will be active in the corresponding graph. The above pair of constraints is non-linear because of the OR function and the feasible region is non-convex. The constraints could have been made linear by a priori choosing one of the constraints and dropping the other. This however will reduce the size of the feasible region and will result in inferior compaction. Figure 2 illustrates a case that gives rise to a single arc constraint and Figure 3 illustrates a case that produces pairs of constraints.

Another type of constraints come from connectivity requirements. In the course of compaction, one has to maintain connections between the lines and the corresponding symbols. A connection point between a line and a symbol does not have to be fixed. It can slide along the edge of the connecting layer, as long as it does not violate line-width or distance rules. The connectivity requirement is coded as a pair of inequalities. This time the pair is coded as two arcs between the same two x nodes or y nodes. For an example, see Figure 4.

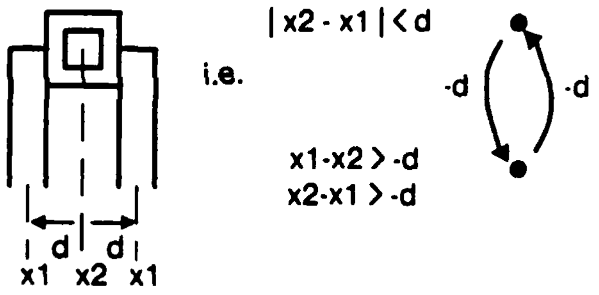


Figure 4: Connectivity constraints

As we shall see later, the connectivity requirements complicate the longest path algorithm we use, since the graph is no longer acyclic.

Problem Formulation

Let us now introduce some notation and definitions:

There are two graphs G_x and G_y that encode the set of constraints, one for horizontal constraints, G_x, and one for vertical constraints, G_y.

$G_x = \{X, E_x, W_x\}$ is a directed graph. $X = \{x_i\}$ is the set of nodes in the graph, one for each symbol instance, one for each vertical line and two special nodes s_x (the source) and t_x (the sink). s_x corresponds to the left boundary of the cell and t_x corresponds to the right boundary of the cell. An arc is put between s_x and all symbols and vertical lines that are visible from the left side of the cell. Similarly arcs are put between all symbols that are visible from the right and the sink node t_x.

$E_x = \{\langle x_i, x_j\rangle \mid x_i, x_j \in X\}$ is the set of arcs in the graph, one arc for each inequality. The set of inequalities E_x is divided into three sets.

$E_x = A_x \cup R_x \cup D_x$.

A_x is the set of simple arcs. These arcs correspond to simple horizontal constraints. R_x is the set of vertical intra-group arcs. The arcs in R are divided into groups. Each vertical intra-group corresponds to a collection of symbol instances that are connected by vertical lines. The arcs in these groups correspond to connectivity constraints. They come in pairs (as was shown in Figure 4). Figure 5 shows an example of a vertical group and the corresponding graph.

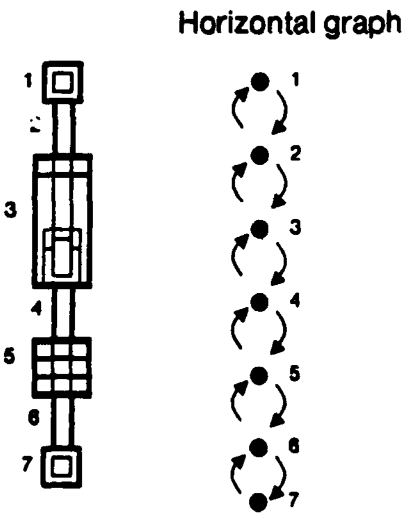


Figure 5: A Vertical Group

D_x is the set of arcs that correspond to horizontal constraints that have a dual vertical constraint. Such constraints are illustrated in Figure 3.

W_x is a set of weights.

$W_x = \{w^x_{ij} \mid \langle x_i, x_j \rangle \in E_x, w^x_{ij} \in \mathbb{R}\}$.

Each arc in the set E_x has a weight attached to it, corresponding to the horizontal distance requirements.

The graph G_y and the sets A_y, R_y, D_y are defined in an analogous way to G_x, A_x, R_x, D_x respectively by interchanging horizontal and vertical axes and changing x to y.

In addition to the two graphs, there is a set of decision variables I. Each decision variable d_i, corresponds to a pair of arcs $\langle e^x_i, e^y_i \rangle$, $e^x_i \in D_x$, $e^y_i \in D_y$. By construction, D_x and D_y have the same number of arcs. Each arc in D_x corresponds to one and only one arc in D_y. Each decision variable can take the values $\{0,1\}$.

The optimization problem:

For each node $x_i \in G_x$ and $y_i \in G_y$, we assign values x_i and y_i respectively. (We use the node name and its value interchangeably, hopefully without confusing the reader).

The vector of values $z = \langle x, y, d \rangle$ is said to be feasible if the following conditions are satisfied:

For each arc $e \in A_x \cup R_x$; $e = \langle x_i, x_j \rangle$

$x_i - x_j \geq w^x_{ij}$

Similarly, for each arc $e \in A_y \cup R_y$; $e = \langle y_i, y_j \rangle$

$y_i - y_j \geq w^y_{ij}$

and for each $d_i \in I$

$x_i - x_j \geq w^x_{ij}$ if $d_i = 1$

$y_i - y_j \geq w^y_{ij}$ if $d_i = 0$

where d_i corresponds to the pair of dual arcs (e^x_{ij}, e^y_{ij}).

Let $\Omega = \{z \mid z = \langle x, y, d\rangle \; ; \; z \text{ is feasible}\}$

be the set of feasible points. Our problem is

Problem 1:

$$[\min_{z \in \Omega} f(x, y) \; .]$$

f(x, y) is the objective function. Ordinarily the objective function is the area occupied by the cell. In that case $f(x, y) = (t_x - s_x)(t_y - s_y)$.

It is possible, however, to manipulate the objective function in order to get certain effects. For example if compacting in the horizontal direction is more desirable than compacting in the vertical one, $f = (t_x - s_x)(t_y - s_y) + C(t_x - s_x)$ with large C could be used.

The compaction and layout generation has been formulated as a graph minimization problem. The task at hand therefore is to translate the stick diagram into the graph form, solve the minimization problem, and then translate the solution into an actual layout. In this section we are going to give some details about the stick diagram to graph problem translation. The algorithm used for solving the graph minimization problem is given in Section 3. Once the minimization problem is solved, the layout generation part is straightforward.

At first sight, the translation from the stick diagram to a graph problem looks easy. Connectivity constraints can be generated by looking at the two ends of each line. The difficulty arises in the distance constraint generation. First, pieces of circuits that are on the same electrical node and are on the same layer do not have to be constrained. This is solved by doing electrical network extraction as part of the translation.

Second, in principle each piece of circuit can interact with any other piece of circuit and therefore a constraint has to be put between the two. In reality, the interaction between different pieces is very local. Although it would not be "wrong" to put constaints between pieces of circuits that do not interact (the solution of the resulting problem will still be correct), clearly it is undesirable to do so. Using more constraints than necessary requires more space and more time for the solution. Finding the smallest number of constraints necessary for the compaction problem seems to be very hard. Our approach is to try and limit the number of extraneous constraints but not necessarily to find the minimal set of constraints. After staring a while at some integrated circuits, one realizes that the main reason that pieces of cirucuits do not interact is the fact that circuits tend to form small compartments. One can think of the pieces of circuits, transistors, lines, and so on as the walls of the compartments and the space in between as the area that makes the compartment. Two pieces of circuit that do not share a compartment could not possibly interact since compaction leaves the topology of the circuit invariant. Since there are no distance restrictions between metal and the other layers (poly, diffusion implant), one needs to construct two kinds of compartments, one for metal and one for the other layers (See Figure 6). A scan-line based "ponds and islands" algorithm is used to identify the compartments and to group the pieces that make the compartment walls.

3. Methods for Solving the Mathematical Problem

In the previous section a detailed description of the graph optimization problem to be solved was given. A full description and analysis of the solution methods and algorithms one could use for solving the above problem cannot be given here; it will be reported elsewhere [Watanabe]. We are, however, going to outline the major ideas and algorithms that are used to solve the problem.

Problem 1 is made of two parts - an integer part and a graph part. Let **d** be a vector of decision variables $\mathbf{d} = \langle d_1, ..., d_{|I|} \rangle$ $d_i \in \{0,1\}$. Clearly **d** can take $2^{|I|}$ different values. For each fixed value of **d**, say d_0, one has two graphs

$g_x(d_0) = \{X, E_x(d_0), W,\}$ and $g_y(d_0) = \{Y, E_y(d_0), W_y\}$. $g_x(d_0)$ is a subgraph of G_x and $g_y(d_0)$ is a subgraph of G_y. The arcs in $g_x(d_0)$ are all the arcs in $A_x \cup R_x$ and the set of arcs in D_x that corresponds to the components of d_0 that are equal to 1. That is,

$$E_x(d_0) = R_x \cup A_x \cup D_x\,(d_0)$$

where $D_x(d_0) = \{e_i \mid e_i \in D_x \text{ and } d_i = 1\}$

Similarly, $D_y(d_0) = \{e_i \mid e_i \in D_y \text{ and } d_i = 0\}$.

A few observations are in order:

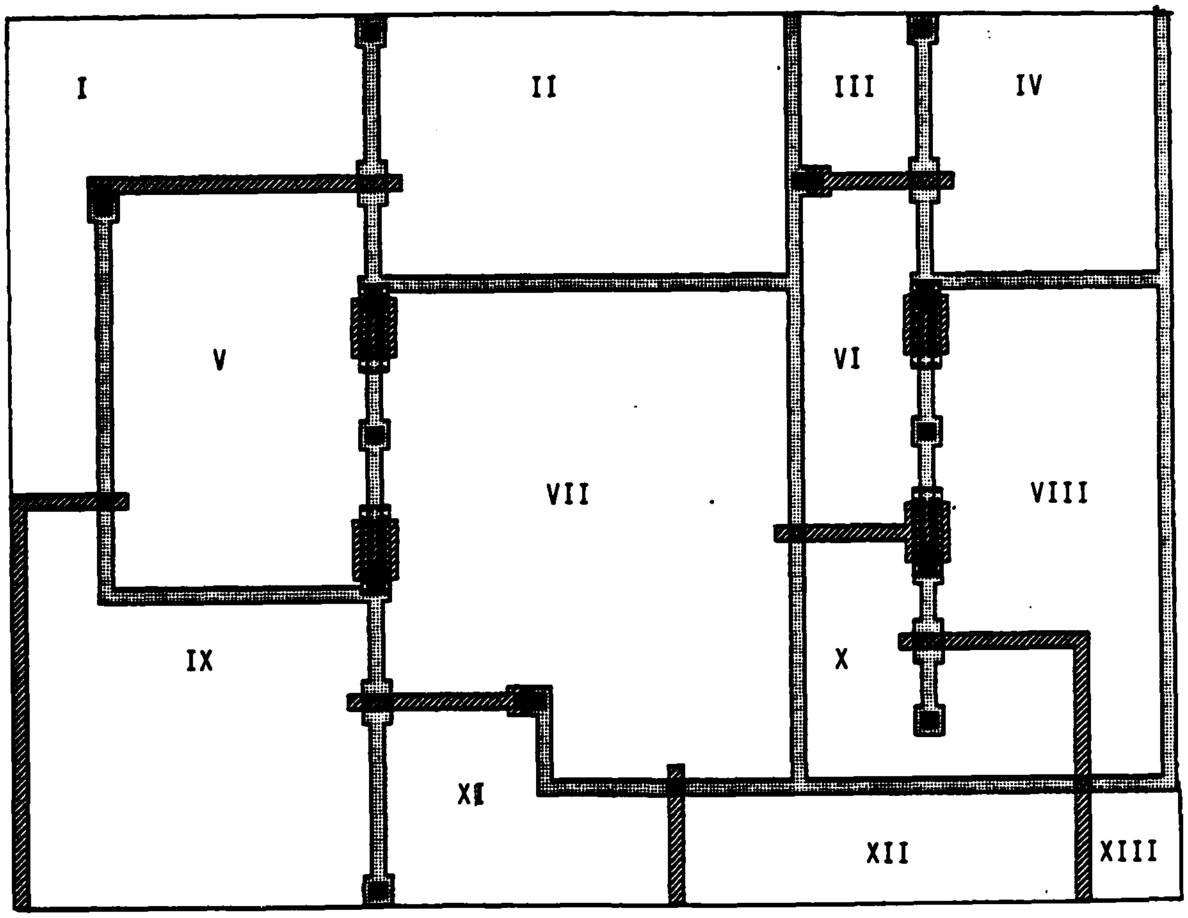


Figure 6: Compartments in T-flipflop circuit

1) For each fixed value of **d** one has two separable graph problems, one for the x-variables and one for the y-variables.

2) Since the objective function is monotone in x and y, each one of the graph optimization problems is simply the longest path algorithm.

3) Although the graphs G_x and G_y are not acyclic, they are "almost" acyclic. Only a simple modification of the longest path algorithm is needed.

4) If $\mathbf{d}_x^1$ is different from $\mathbf{d}_x^2$ only in one component, say $d_i^1 = 1$ and $d_i^2 = 0$, then $g_x(\mathbf{d}^1)$ is different from $g_x(\mathbf{d}^2)$ only by one arc. Namely, $g_x(\mathbf{d}^2)$ is obtained from $g_x(\mathbf{d}^1)^2$ by removing the arc that corresponds to the decision variable d_i. Similarly $g_y(\mathbf{d}^2)$ can be obtained from $g_y(\mathbf{d}^1)$ by adding the arc that corresponds to d_i. Therefore, if one has solutions for the longest path problems for $g_x(\mathbf{d}^1)$ and $g_y(\mathbf{d}^1)$, finding the longest path solution for $g_x(\mathbf{d}^2)$ and $g_y(\mathbf{d}^2)$ could be done by updating the longest path solution at hand with fewer steps than otherwise will be required.

The algorithm used to find the best decision variables combination is "branch and bound" (see Garfinkel and Nemhauser [1972]). The "branch and bound" technique is basically an exhaustive search by enumeration. The set of all possible values of the decision variables can be thought of as a complete binary decision tree. The leaves of the decision tree correspond to a fixed set of values of the decision variables. Branch and bound technique is a depth first search for the best answer. At each stage of going down the decision tree, one more 0-1 variable is being fixed. The main idea behind branch and bound is to compute at each stage a global upper bound and a local lower bound for the subtree that is searched. Each upper bound is a feasible solution. The lower bound is not. The moment a lower bound exceeds the best upper bound that was found so far, the corresponding subtree can be ignored since no feasible solution for that subtree will improve the global result.

Each lower bound is computed by dropping the constraints that correspond to decision variables that have not been fixed yet. An upper bound is computed by picking a specific value for the decision variables that agrees with the set of decision variables that have been fixed.

Branch and bound is a standard algorithm and will not be described here. However, these observations are in order:

1) The algorithm starts by computing an initial guess. The better the initial guess, the better branch and bound performs. We use a heuristic method for finding the initial guess. It has been observed that the initial guess yields a solution that is about 10% worse than the optimal one.

2) At each step of the branch and bound, an arc from the critical path of g_x and g_y is removed. Therefore, if some decision variable corresponds to arcs that do not become part of a critical path, that decision variable never becomes part of the branch and bound tree.

3) The choice of where to branch next is arbitrary. We use a heuristic to decide where to branch. In all cases tested so far, the optimal solution was found after few branchings (less than 100). However, verifying that a solution is optimal took a long time. How exhaustive a search to make, however, could be set by the user.

4) By modifying the pruning procedure of the branch and bound algorithm, it is possible to speed up the search tremendously but at the expense of perhaps finding only a suboptimal solution. In our experience, even the initial solution is a good suboptimal solution.

4. Implementation, Examples, Conclusions

An experimental program to test our ideas was constructed. The program is written in PASCAL. The input to the program is written in CIF and the output from the program is a CIF file. The program compacts NMOS stick diagrams using Mead & Conway design rules. A decision was made not to provide a fancy user interface. However, using CIF as the input/output language allowed us to use existing tools. Although the program works only with Mead & Conway design rules for NMOS, it can easily be adapted to other technologies and different design rules. Most of the technology and process-dependent rules are in table form. Also, there are no restrictions on the predefined library of cells. That set can have any user defined cells.

Our initial experience with the program has been very encouraging. First, the method used for constructing an initial guess yields very good results. In all cases tested, the initial guess was within 10% of the optimal solution. Second, the heuristics used in the branch and bound search worked very well. Even though the number of decision variables can be large (100-1000 or more), the optimal solution is found quite fast, usually after less than 100 steps. Verifying that the solution is optimal, however, can be very expensive. Thousands of branch and bound steps may be necessary. To date, a good stopping strategy, except limiting the number of branch and bound steps, has not been found. Automatic insertion of jog points has not been implemented as yet.

To illustrate the quality of compaction achieved, two "before and after" examples are given. The first example is a T flipflop. The input is given in Figure 1. The result is given in Figure 7. The optimum was found after 8 steps and it was verified at the 10th step. The program, running on a VAX 11/780 under UNIX, took 11.8 seconds to produce the answer.

The second example, Figures 8a - 8b, is a priority queue cell [Kedem 1981]. The optimal answer was found after 48 steps. However, thousands of steps were required to verify that the solution was optimal. It might be of interest to note that the cell was first laid out by hand. The result produced by the program was better than the hand-crafted one. It took the program about 2 minutes, 11.4 seconds to find the final answer, 42.2 seconds for preprocessing, 50.9 for finding first guess, and 38.9 seconds for branch and bound.

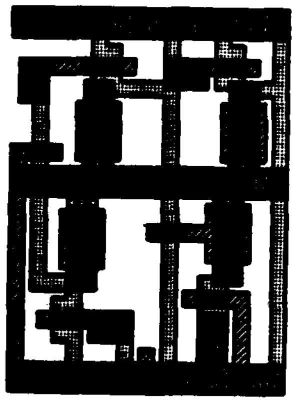
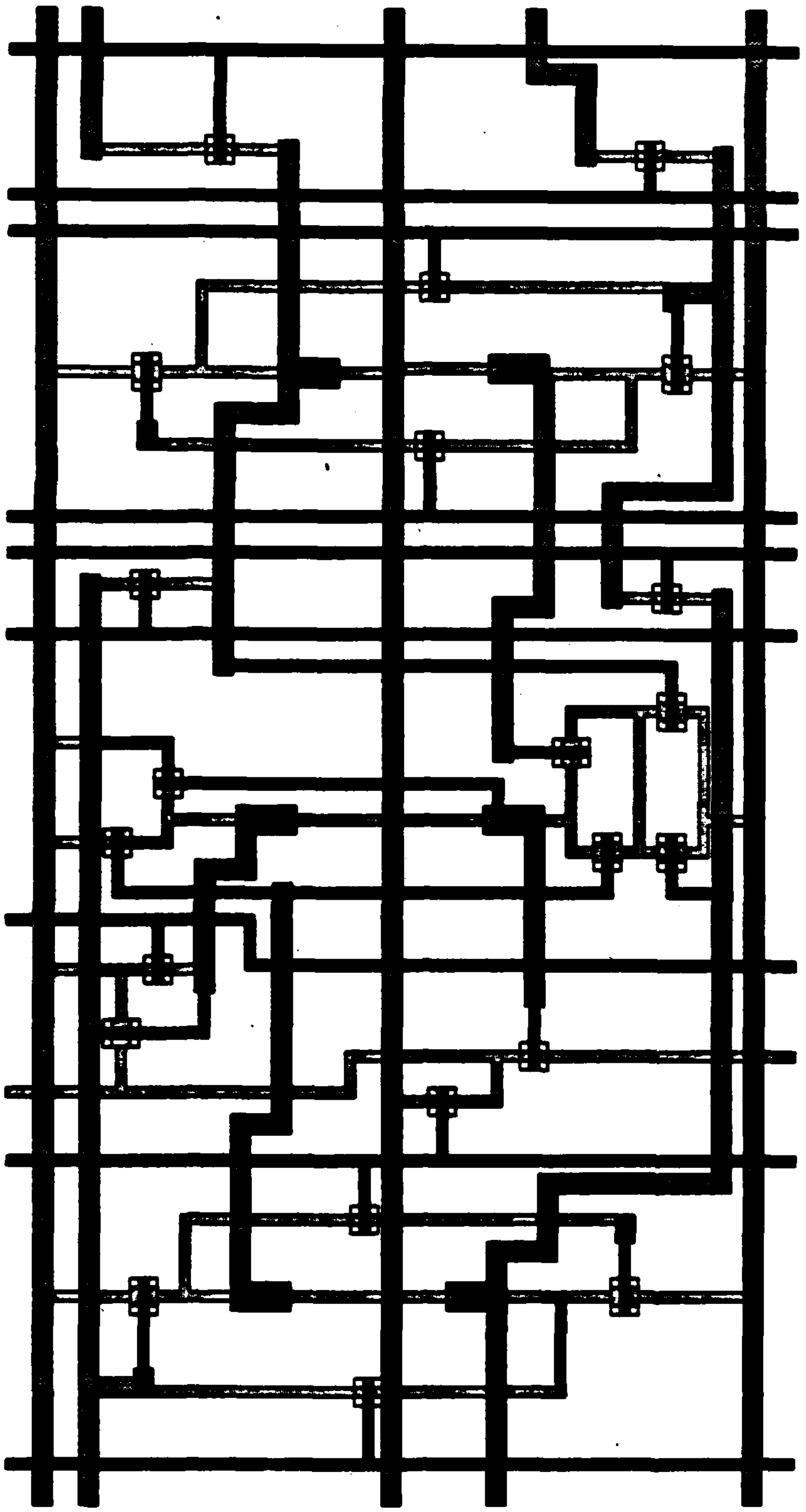


Figure 7: T-flipflop compacted


8A

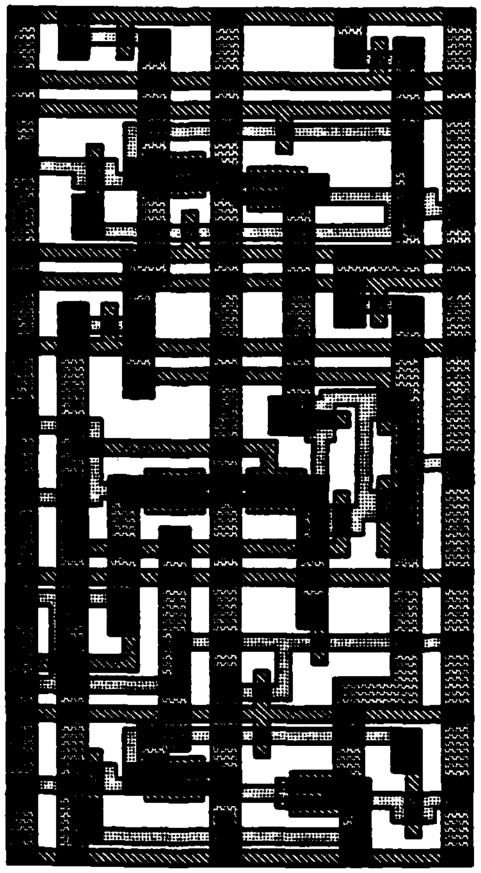

8B

Figure 8a (left): Priority Queue Cell Stick Diagram
8b (right): Priority Queue Cell Compacted

References

Dunlop, Alfred E., "SLIP: Symbolic Layout of Integrated Circuits with Compaction", *Computer Aided Design*, Vol. 10, No. 6, 387-391, November 1978.

Dunlop, Alfred E., "Integrated Circuit Mask Compaction", Ph.D. Thesis, Department of Electrical Engineering, Carnegie-Mellon University, October 1979.

Dunlop, Alfred E., "SLIM-The Translation of Symbolic Layouts into Mask Data", *Proceedings*, 17th Design Automation Conference, 595-602, Minneapolis, Minn., June 1980.

Garfinkel, R. S. and Nemhauser, G. L., *Integer Programming*, John Wiley & Sons, Inc., 1972.

Hachtel, G. D., "On the Sparse Tableau Approach to Optimal Layout", *Proceedings, IEEE ISCAS*, 1019-1022, 1981.

Hsueh, Min-Yu, "Symbolic Layout and Compaction of Integrated Circuit", Memorandum No. UCB/ERL M79/80, Electronics Research Lab., University of California Berkeley, December 1979a.

Hsueh, Min-Yu and Pederson, Donald O., "Computer-Aided Layout of LSI Circuit Building-Blocks", *Proceedings*, 1979 International Symposium on Circuits and Systems, 747-477, Tokyo, Japan, 1979b.

Kedem, Gershon, "A First In, First Out and a Priority Queue", TR90, Department of Computer Science, University of Rochester, March 1981.

Lawler, Eugene, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, 1976.

Mead, C. and Conway, L., *Introduction to VLSI Systems*, Addison-Wesley, 1979.

Mosteller, R. C., "REST User Guide", SSP File #4030, Computer Science Department, California Institute of Technology, October 1980.

Ohtsuki, T., Sugiyama, N. and Kawanishi, H., "An Optimization Technique For Integrated Circuit Layout Design", *Proceedings*, Kyoto International Conference on Circuit and System Theory, 67-68, Kyoto, Japan, 1970.

Otten, R. H. J. M., and van Lier, M. C., "Automatic IC-Layout: The Geometry of The Islands", *Proceedings*, 1975 IEEE International Symposium on Circuits and Systems, 231-234, Newton, Mass., April 1975.

Preas, B. T., "Methods for Hierarchical Automatic Layout of Custom LSI Circuit Masks", Ph.D. Thesis, Department of Electrical Engineering, Stanford University, August 1979a.

Preas, B. T. and Gwyn, C. W., "General Hierarchical Automatic Layout of Custom VLSI Circuit Masks", *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 3, No. 1, 41-58, January 1979b.

Rowson, Jim, "Understanding Hierarchical Design", Ph.D. Thesis, Computer Sceince Department, California Institute of Technology, April 1980a.

Rowson, Jim, "Geometry Composition -- Another Look", SSP File #3792, Computer Science Department, California Institute of Technology, June 1980b.

Trimberger, Stephen, "The Proposed Sticks Standard", Technical Report 3880, Computer Science Department, California Institute of Technology, October 1980.

Williams, John D., "STICKS--A New Approach To LSI Design", Master's Thesis, Massachusetts Institute of Technology, June 1977.

Williams, John D., "STICKS--A Graphical Compiler for High Level LSI Design", *AFIPS Conference Proceedings*, Vol. 47, 289-295, June 1978.

Watanabe, Hiroyuki, "IC Layout Compaction Using Mathematical Optimization", Ph.D.Thesis, Department of Computer Science, University of Rochester, In preparation.

END

FILMED

9-83

DTIC